MTcast: Robust and Efficient P2P-based Video Delivery for Heterogeneous Users

Tao Sun, Morihiko Tamai, Keiichi Yasumoto, Naoki Shibata †,
Minoru Ito and Masaaki Mori †

Graduate School of Information Science, Nara Institute of Science and Technology,
Ikoma, Nara 630-0192, Japan
e-mail: {song-t,morihi-t,yasumoto,ito}@is.naist.jp
† Department of Information Processing and Management, Shiga University,
Hikone, Shiga 522-8522, Japan
e-mail: {shibata,mori}@biwako.shiga-u.ac.jp

Abstract. In this paper, we propose a new video delivery method called *MTcast (Multiple Transcode based video multicast)* which achieves efficient simultaneous video delivery to multiple users with different quality requirements by relying on user nodes to transcode and forward video to other user nodes. In MTcast, each user specifies a quality requirement for a video consisting of bitrate, picture size and frame rate based on the user's environmental resource limitation. All users can receive video with the specified quality (or near this quality) along a single delivery tree. The main characteristics of MTcast are in its scalability, high user satisfaction degree in received video quality, short startup latency and robustness against node failure. Through simulations, we have confirmed that MTcast can achieve much higher user satisfaction degree and robustness against node failure than the layered multicast method.

Keyword:video multicast, transcode, QoS, service overlay networks

1 Introduction

There is a demand for an efficient video delivery method for *heterogeneous user nodes* which have different computation powers, display sizes and available bandwidths. There are several approaches for simultaneously delivering video to multiple users with different quality requirements. In the multiversion technique [1], multiple versions of a video with different bitrates are prepared in advance so that the best one can be delivered to each user, within resource limitation. In the online transcoding method [2], an original video is transcoded at a server or an intermediate node (i.e. proxy) to videos with various quality, according to receivers' preferences, and forwarded to the receivers. In the layered multicast method [3, 4], video is encoded with layered coding techniques such as [5] so that each user can decode the video by receiving arbitrary number of layers. Since each layer is delivered as an independent multicast stream, each user can receive as many layers as possible within his/her resource limitation. In this method, as the number of users increases, more layers are required in order to improve user satisfaction degree. However, decoding video from many layers consumes large processing power and buffers. In [3], a method for optimizing bitrate of each layer to maximize user satisfaction degree is proposed. In the multiversion method, the control mechanism is

simple, but not efficient in terms of server storage and network bandwidth usage. In the multiversion and layered multicast methods, there can be a large gap between the requested quality and the delivered quality if there are not enough number of versions or layers. The online transcoding method can satisfy all the above requirements since it can transcode original video to arbitrary quality video. But, large computation power required for transcoding can be a problem.

There are many studies on video streaming in peer to peer networks. [6] has proposed the Overlay Multicast Network Infrastructure (OMNI). In OMNI, each user node works as a service provider as well as a service user, and a multicast tree is composed of user nodes so that the video delivery service is provided to all the user nodes through the tree. OMNI can adapt to the change of the user node distribution and the network conditions. [7] has proposed CoopNet where traditional client-server based streamings are augmented when the load of the video server exceeds it's limit. In CoopNet, user nodes cache parts of stream data, and deliver them through multiple diverse distribution trees to the user nodes while the server load is high. OMNI and CoopNet aim at adapting the video delivery service depending on the dynamic change of network conditions, server load and so on. However, they do not treat video delivery to user nodes with different quality requirements.

We propose a new video delivery method called *MTcast (Multiple Transcode based video multicast)* which achieves efficient simultaneous video delivery to multiple heterogeneous users by relying on user nodes to transcode and forward video to other user nodes. In MTcast, each user specifies a quality requirement for a video consisting of bitrate, picture size and frame rate based on the user's environmental resource limitation. All users can receive video near specified quality along a delivery tree. Each user can change the quality requirement each time segment or each video shot.

We have considered the following criteria : (1) *high scalability* for accommodating a large number of users, (2) *high user satisfaction* in the sense that the delivered quality is close to the required quality, (3) *small resource consumption* within available resource of each user node, (4) *short startup latency* to start playing back video quickly, (5) *reasonable number of transcoding times* for keeping good video quality as well as short delivery latency, and (6) *high robustness* for continuing video delivery service even with node/link failures.

In order to achieve the above (1) to (3), a delivery tree called *transcode tree* whose root is the sender of a video content, is constructed as a perfect n-ary tree, where user nodes with higher quality requirements are located near the root of the tree, and nodes with lower quality requirements are located far from the root. Nodes are placed according to their computation power, available downstream and upstream bandwidths. Each node in the tree receives a video stream, transcodes it to lower quality video in real time and forwards it to its children nodes. In order to achieve the above (4) to (6), nodes are grouped so that each group has k members with similar quality requirements. These groups are called *layers*. All nodes in a layer receives the video with the same quality from their parent nodes along the transcode tree. We let the representative node of each layer keep the complete information of the tree. This allows a new receiver to easily find the layer which has the closest quality to its own quality requirement and to quickly send a request to the node in the layer to start delivery of the video. In order to accommodate new receivers or to replace faulty nodes with normal ones, we made each layer to keep a certain amount of extra computation power and available upstream bandwidth (computed from the value of k). In general, if we use a large number for k,

we can improve performance of the above (4) to (6). However, user satisfaction degree may be reduced since the received video quality is averaged over k members. So, in the proposed method, we adopted an approach to dynamically increase the value of k as the total number of receivers increases. When the number of receivers is sufficiently large, we can make both user satisfaction and system robustness high.

After certain time elapses, extra resources at a layer might have been exhausted. So, our method reconstructs the transcode tree periodically or at each time boundary between subsequent video shots. When video delivery requests and failures occur after extra resources of a layer have been exhausted, they are processed during the next tree reconstruction.

We have investigated performance of MTcast by simulations using topologies generated by Inet3.0[8]. As a result, we have confirmed that MTcast can achieve both higher user satisfaction degree and higher robustness than the layered multicast method.

2 Target Environment

In this paper, we deal with a method for simultaneously delivering a video content to multiple *heterogeneous users* who have different available bandwidth, different computation power, and different display resolutions. Here, we assume the following types of user terminals, types of communication infrastructures and target contents. **user terminal:** desktop PC, laptop PC, PDA, cellular phone, etc. **communication infrastructure:** either fixed broadband (leased lines, ADSL,CATV, etc.) or wireless network (wireless LAN, W-CDMA,Bluetooth, GSM/PDC, etc). **the total number of users:** 500 to 100,000. **target contents:** video (both recorded and live).

We assume a service which starts to transmit a video content to all receivers at the same starting time like TV broadcast. Even after the starting time of the video, users can start to receive the video anytime, but the video can be watched from the scene currently in transmission.

We assume that user nodes are connected to each other through overlay links, and that each node uses overlay multicast to transmit/receive streams to/from the other node.

In the multicast tree, we let each user node except leaf nodes transcode a video stream and forward it to its children nodes as well as receive and play back the stream.

From the above discussion, the main purpose of this paper is to build and manage the multicast tree which satisfies criteria (1) to (6) in Sect. 1 and to devise the efficient video delivery method using the tree.

3 MTcast

In this section, first we briefly define notations used in our MTcast algorothm, and then explain the details of MTcast.

3.1 Definitions

Let s denote a video server, and $U = \{u_1, ..., u_N\}$ denote a set of user nodes. We assume that for each $u_i \in U$, available upstream (i.e., node to network) bandwidth and downstream (i.e., network to node) bandwidth are known in advance. We denote them by $u_i.upper\_bw$ and $u_i.lower\_bw$, respectively. Let $u_i.q$ denote u_i's video quality requirement. In general, as $u_i.q$, multiple video parameters such as bitrate, picture size and frame rate are specified. In this paper, we assume that $u_i.q$ represents only bitrate

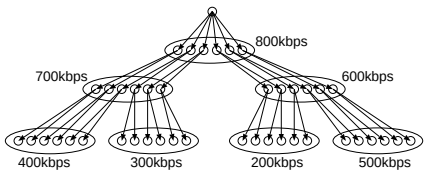

Fig. 1. Example of Transcode Tree, where $n = 2, k = 6$

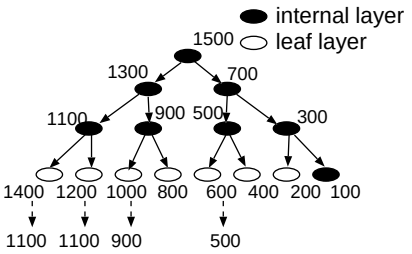

Fig. 2. Tree construction in order of depth-first search

of video. [1] Let $u_i.n_{trans}(q)$ denote the maximum number of simultaneous transcoding which can be executed by u_i for videos with quality q. Let $u_i.n_{link}(q)$ denote the maximum number of simultaneous forwarding of videos with quality q which can be performed by u_i. $u_i.n_{trans}(q)$ and $u_i.n_{link}(q)$ are calculated from computation power of u_i, $u_i.upper\_bw$ and video quality.

In the proposed method, we construct a multicast tree where s is the root node and user nodes in U are intermediate (internal) or leaf nodes. Hereafter, this multicast tree is called the *transcode tree*.

3.2 Structure of Transcode Tree

Internal nodes in the transcode tree transmit a video stream to children nodes. In the proposed method, we assume that fanout (degree) of each node is basically a constant (denoted by n). As we will explain in Sect. 3.3, we decide the value of n depending on available resources of user nodes.

In order to reduce the number of transcoding between the root node and each leaf node, we construct the transcode tree as a modification of complete n-ary tree where degree of the root node is changed to k instead of n (k is a constant, explained later). In the transcode tree, for each node $u_i \in U$ and each of its children nodes u_j, $u_i.q \geq u_j.q$ holds.

In order to tolerate node failures and to shorten startup delay of video delivery, every k nodes in U are bunched up into one group. We call each group a *layer*, where k is a predetermined constant, as shown in Fig. 1. We let user nodes in the same layer receive video with the same quality. This quality is called the *layer quality*. A representative node is selected for each layer. Parent-child relationship among all layers on the transcode tree is called the *layer tree*.

An example of the transcode tree with $n = 2$ and $k = 6$ is shown in Fig. 1. ere, small circles and big ovals represent nodes and layers, respectively. Each bitrate (e.g., 500kbps) represents the layer quality.

3.3 Construction of Transcode Tree

In our method, the transcode tree is calculated in a centralized way by one of the nodes. The way of deciding the calculation u_c is explained later. We assume that u_c has information of a video server s and user nodes $U' \subseteq U$ who have requested video. Our tree construction algorithm consists of the following three steps.

Let $A = \{s\} \cup U'$, denote the set of all nodes. In the first step, our algorithm divides A into the set of candidate internal nodes U_I and the set of leaf nodes U_L. We always put s into U_I.

$$u.n_{trans}(u.q) \geq 1 \tag{1}$$

$$u.n_{link}(u.q) \geq n + 1 \tag{2}$$

[1] A method to treat a parameter vector as quality is discussed in [9].

For each node $u \in A$, the algorithm checks if the above inequalities hold or not. If they hold for u, then u is put into U_I, otherwise put into U_L. The above inequalities (1) and (2) represent whether node u can perform transcoding of one or more videos and whether u can forward $n+1$ video streams.

After that, if $|U_I| < \frac{1}{n}|A|$, quality requirements of $|U_L| - \frac{n-1}{n}|A|$ nodes in U_L with larger upstream bandwidths are reduced so that the inequalities (1) and (2) hold. Then, those nodes are moved to U_I. By the above procedure, $|U_I| \geq \frac{1}{n}|A|$ always holds.

In the second step, the algorithm assigns the set of all nodes A to layers. Elements of U_I are sorted in decreasing order of their quality requirements and every bunch of k elements is packed to an internal layer. Here, we select the first node of each layer as the representative node of the layer. The average value of quality requirements is assigned as the layer quality. For the set of leaf nodes U_L, elements are similarly packed to leaf layers.

In the last step, the transcode tree is constructed. The algorithm sorts internal layers in decreasing order of layer quality, and constructs the complete n-ary tree of those internal layers so that the layer quality of each layer does not exceeds that of its parent layer. Next, the algorithm attaches each leaf layer L to the internal layer whose layer quality is closest to L. If the layer quality of L exceeds that of L's parent layer, the layer quality of L is adjusted to that of L's parent. As for the order of assigning internal layers to n-ary tree as shown in Fig. 2.

Finally, the transcode tree is obtained by assigning internal nodes and leaf nodes to internal layers and leaf layers in decreasing order of their required quality, respectively.

Adaptation to available bandwidth between nodes In our method, after constructing the layer tree, each node which belongs to the child layer selects an actual delivery node from k nodes in the parent layer. Whether each child node can receive the video with the requested quality or not depends on the available bandwidth on the path, that is, links on a physical network connecting the child node to the parent node. Below, we describe how to decide the parent nodes by taking into consideration of the physical topology of the network and available bandwidths on paths in the network. Here, we also consider the case that two or more overlay links share the same physical links and thus compete the available bandwidths on those links.

Let C and P be the sets of nodes which belong to a layer and its parent layer, respectively. We suppose that, for each pair of nodes between the child layer and the parent layer, the physical path and the available bandwidth can be obtained with tools such as *traceroute* and *pathload*[10], respectively. Let $bw(c,p)$ and $L(c,p)$ denote the available bandwidth measured with a tool like pathload (called *measured available bandwidth*, hereafter) and the set of links between $c \in C$ and $p \in P$ except for links connected to nodes c and p, respectively. Next, we estimate the worst-case available bandwidth of each overlay link (called *estimated available bandwidth*, hereafter) by considering some of links are shared among multiple overlay links. Initially, for each pair of nodes $(c,p) \in C \times P$, the estimated available bandwidth of each link $l \in L(c,p)$ is set to $bw(c,p)$. The estimation is done based on the *link stress* of each link (i.e., the number of overlay links which use the same physical link for the same data transmission) as follows. (1) The initial link stress is set to 0 for each physical link. (2) For each pair $(c,p) \in C \times P$ and for each link $l \in L(c,p)$, the link stress of l is incremented. However, once the link stress has been already incremented by node c, we do not let other paths including c increment the link stress of the same link to avoid duplicated counting. Based on the measured available bandwidth and the link stress of each physical link,

we decide the parent node of each child node as follows. (i) For each $c \in C$, the following step (ii) is examined in increasing order of node ID. (ii) For each $p \in P$, whether node p can deliver the video with the specified bitrate to node c or not is decided based on the estimated available bandwidth on path $L(c, p)$. If there is no parent node which has enough available bandwidth for the video delivery to node c, node c is moved to a lower quality layer. If only a node can deliver video to c with required bitrate, this node is selected as the parent node of c, and the following step (iv) is executed. If there are multiple nodes which can deliver video to node c with the required bitrate, the following step (iii) is applied to selecting the parent node of c. (iii) For each node $p \in P$ which can deliver video to c with the required bitrate, the new estimated available bandwidth for each link in $L(c, p)$ is calculated by dividing the current estimated bandwidth by the link stress. One node with the largest estimated available bandwidth is selected as the parent node of c. (iv) Once node p is selected as the parent of c, we re-calculate the link stress of each link $l \in L(c, p)$ without incrementing it by the paths including c and subtract the bitrate of the video from the estimated available bandwidth of l. If some bandwidth is still remaining in l, it can be used for another overlay links.

We compared our bandwidth adaptation method with hop count first method where each node greedily selects a parent node which has the minimum hop count. From experiment, we confirmed that our bandwidth adaptation method can achieve higher successful rate (≈ 1.0) of finding parent node which has enough bandwidth to stream video than that of the hop count first method (≈ 0.65) in the similar environment described in sect. 4.

How to decide appropriate values of n and k In our method, the transcode tree is constructed as a modified complete n-ary tree. So, as the value of n becomes large, the tree height (i.e., the number of transcoding) also decreases. Since the required upstream bandwidth of each node increases in proportion of n's value, the value of n must be carefully decided considering upstream bandwidth limitation of each node. We can decide the maximum value of n so that the number of nodes satisfying inequality $u.n_{link}(q) \geq n + 1$ is around to $\frac{1}{n}|A|$. If f nodes may leave from a layer at the same time before the transcode tree is reconstructed, the remaining $k - f$ nodes in the current layer must transmit video streams to $n \cdot k$ children nodes. So, the following inequalities must hold in order to recover from f simultaneous failures in each layer.Thus, the appropriate value of k can be decided from values of n and f.

$$(k - f)u.n_{link}(q) \geq n \cdot k \wedge (k - f)u.n_{trans} \geq \lceil \frac{k}{u.n_{link}(q)} \rceil n \tag{3}$$

3.4 Behavior of MTcast

Startup Behavior Let t denote the time of video delivery. Each user who wants to receive video stream sends a video delivery request to the video server s before time $t - \delta$. At time $t - \delta$, s calculates the transcode tree with the algorithm explained in Sect. 3.3. Here, δ is the time to calculate the transcode tree and distribute the necessary information to all nodes. s also decides the node u_c which calculates the transcode tree next time. u_c is selected from representative nodes of layers which have sufficient downstream bandwidths. Next, s distributes the information which is necessary for video delivery to all nodes in T.

For information distribution, s composes data I which contains the complete information on T, its layer tree, representative nodes and quality of layers, and u_c. Then, it

sends I to the representative node of the root layer. Then, the node forwards the information to its children layers' representative nodes. Data I is propagated until all leaf layers' representative nodes receive it. When each representative node receives the data I, it sends part of the information in I to member nodes of the same layer. We let each representative node keep (a) the whole layer tree with each layer's layer quality and representative node's address, and (b) its responsible layer and addresses of the layer's member nodes. We also let each node keep (1) addresses and layer quality of children nodes, (2) current layer's quality and responsible node's address, (3) parent node's layer and its responsible node's address, and (4) node u_c to calculate the transcode tree next time. By the above steps, information of the transcode tree is shared among all nodes and video gets ready to be delivered.

How to cope with new delivery requests and node failures As explained in Sect. 3.3, each node in an internal layer has an extra upstream bandwidth for forwarding one more video stream. A user node u_{new} who has requested video delivery after time t can use this extra bandwidth to receive a video stream. Here, the fanout of the forwarding node u_f which sends a stream to u_{new} is allowed to be $n+1$ tentatively. The forwarding node u_f does not need to transcode a video stream for u_{new}, since u_f is already transmitting a video stream to n children nodes and it transmits the same stream to u_{new}.

If one or more nodes in a layer fail or suddenly leave from the transcode tree, all of their descendant nodes will not be able to receive video streams. Our method allows children nodes of the failure nodes to find alternative nodes in the same layer as those failure nodes and to ask them to forward video streams. Therefore those alternative nodes use their extra upstream bandwidths similar to the case of processing new delivery requests.

As we will explain later, the transcode tree is reconstructed periodically, the fanout of each stream is reduced to n or less than n and the consumed extra upstream bandwidth is regained after reconstruction.

If the representative node of a layer fails, children nodes of the representative node cannot find new parent nodes. Thus, one of other nodes in the layer becomes sub representative node, and nodes in children layers keep address of these nodes. When the representative node fails, one of children nodes of the representative node sends switch request to the sub representative node so that the sub representative nodes becomes the new representative node. If a sub representative node fails before the representative node fails, one of other nodes become sub representative node.

Procedure for new delivery requests We assume that a new user node u_{new} knows at least one node u^* in the transcode tree which is already receiving a video stream. u_{new} tries to find the best node in the transcode tree which can be u_{news}'s parent node in the following procedure. (1) u_{new} sends a query with its quality requirement $u_{new}.q$ and its address to u^*. (2) If u^* is not a responsible node of any layer, it forwards the received query to the responsible node u_r of u^*'s current layer. (3) When u_r receives the query, it sends the information of the layer tree to u_{new}. (4) When u_{new} receives the layer tree, it finds the layer which has the layer quality closest to $u_{new}.q$ and sends a video delivery request to the responsible node u'_r of the layer. (5) u'_r selects a node u' and forwards the request to u' which has the required extra upstream bandwidth. (6) Finally, u' starts to deliver a video stream to u_{new}.

Recovery from node failure We let each node u monitor status of data receiving in real-time, and u thinks that node failure happened when it does not receive any data (or

the average data reception rate is much less than the expected one) during a specified time period. When u detects failure of its parent node u_p, u sends a video forwarding request to the representative node of u_p's layer. Then, similarly to the case of a new video delivery request, the video stream is forwarded from an alternative node if it has an extra upstream bandwidth. At u, video can be played back seamlessly by buffering certain time of video data during the above switching process.

Reconstruction of Transcode Tree User node u_c reconstructs the transcode tree in the following steps. We assume that all nodes know the time t_r when the reconstructed transcode tree is in effect.

Before time $t_r - \delta'$, each node u sends a new quality requirement which will be effective after t_r to the representative node of $u's$ current layer, if u wants to change video quality. Here, δ' is the time to gather quality requirements from all nodes, calculate the transcode tree and distribute the necessary information to (part of) nodes. When the representative node u_L of each layer L receives quality requirements from all members of L and those from representative nodes of L's children layers (if L has children layers), u_L sends the unified list of quality requirements to L's parent layer's representative node. Finally, the representative node of the root layer sends the received list of quality requirements to node u_c. Finally, u_c has quality requirements of all nodes which will be effective after time t_r.

Then, node u_c calculates the transcode tree with the algorithm in Sect. 3.3 and distribute to all nodes the information for the new transcode tree and the node u'_c which calculates the tree next time, as explained in Sect. 3.3.

At time t_r, all nodes stop receiving streams from current parent nodes and the nodes in the root layer of the new transcode tree starts to deliver video streams. Nodes in internal layers also forward video streams after receiving them. The video stream transmitted along the new transcode tree arrives after a certain time lag due to transcode and link latency. So, during the time lag, each node plays back video from its buffer to avoid blank screen.

For the next reconstruction of the transcode tree, the buffer of each node must be filled with video data of the above time lag. This process is done by transmitting the video stream slightly faster than its playback speed. This fast transmission requires more computation power for transcode and more bandwidth for forwarding video data. Let α denote the ratio of the above time lag over the time period between two subsequent tree reconstructions. Here, α is a real constant number between 0 and 1. Then this fast transmission requires computation power and upstream/downstream bandwidths $(1 + \alpha)$ times as much as the normal transmission.

Reconstruction of the transcode tree may greatly change positions of nodes in the tree. So, we let nodes closer to the root node play back video with larger delay by buffering certain time of video data. Data amount to be buffered can be decided with statistic information calculated from received video streams.

4 Evaluation

In order to show usefulness of MTcast, we have conducted several experiments for measuring (1) overhead of transcode tree construction and (2) the user satisfaction degree on received quality.

4.1 Overhead of Tree Reconstruction

In our method, the transcoding tree is reconstructed periodically and/or when a new video segment starts. The overhead of the tree reconstruction consists of (i) aggregation of quality requirements for the new video segment from (part of) user nodes, (ii) calculation of the new transcode tree, and (iii) distribution of the new transcode tree to representative nodes of all layers.

For the above (i), even when the number of nodes is 100,000 [2] and each node sends a 50 Byte packet for quality requirement directly to the computation node u_c, 5 MByte information is sent to the node u_c which computes the transcode tree. If we assume that this information is sent in 10 seconds (it should be less than the period of the tree reconstruction), the average transmission speed becomes 4 Mbps. Since only the node with enough downstream bandwidth can be selected as u_c, this would not be a bandwidth bottleneck.

In order to investigate the impact of the above (ii) and (iii), we measured the size and the computation time of the transcode tree with the number of nodes from 1,000 to 100,000. Here, we assumed that $n = 2$ and $k = 5$, where n and k are the fanout of each internal node and the number of layer members, respectively. The experimental result is shown in Table 1. According to Table 1, the computation time was within 2 seconds even when the number of nodes is 100,000 (Pentium 4 2.4GHz with 256MB RAM on Linux2.6.10). So, computation time would not be a bottleneck.

Table 1. Size and Computation Time of Transcode Tree

number of nodes	computation time (sec)	size of tree (byte)
1000	0.016	3K
10000	0.140	30K
100000	1.497	300K

The size of the transcode tree was 30 Kbyte when the number of all nodes is 10,000. The information of the tree is sent to representative nodes of all layers along the layer tree. If we assume that this is sent in 10 seconds, each representative node needs 24Kbps extra bandwidth. Even when the number of nodes is 100,000, the required bandwidth would be 240Kbps. Also, the tree size can be further reduced with the general compression algorithm like `gzip`.

4.2 User Satisfaction

In this section, we compare MTcast with the layered multicast method in terms of the user satisfaction degree for the quality requirements.

Similarly to [3], the satisfaction degree of user u ($0 \leq S_u \leq 1$) is defined as follows.

$$S_u = 1 - \frac{|u.q - u'.q|}{u.q} \tag{4}$$

Here, $u.q$ represents u's required quality and $u.q'$ represents the quality of the received video. When $u.q'$ is closer to $u.q$, S_u gets closer to 1.

[2] This number is actually much smaller since only the nodes which want to change their quality requirements for the next video segment send the messages.

The experiment has been conducted as follows: The physical network topology with 6000 nodes is generated with Inet3.0 [8] and 1000 nodes are selected as user nodes. Links directly connected to those user nodes are regarded as LANs. Links attached to LAN links are considered as MAN links, and other links are considered as WAN links. We assume that there are the following four types of user nodes: (1) user nodes with cell phone networks whose available downstream bandwidths are 100 to 500 Kbps; (2) user nodes with wireless LAN (2 Mbps to 5 Mbps); and (3) user nodes with wired broadband networks (10 Mbps to 20 Mbps). We assume that each user node has the same amount of available upstream bandwidth as the downstream bandwidth.

Table 2. Configuration of Available Bandwidth

	100k to 500k	2M to 5M	10M to 20M
case1	33%	33%	33%
case2	5%	33%	62%
case3	45%	10%	45%
case4	62%	33%	5%

Table 3. Relationship of $u.n_{transcode}, u.n_{link}, f$

	$p.n_{transcode}$	$p.n_{link}$	k	f
pref. 1	2	4	2	1
pref. 2	1	3	3	1
pref. 3	1	3	6	2
pref. 4	1	3	9	3

We selected the quality requirement of each user node according to one of the following three distributions within the available bandwidth: (a) uniform distribution from 300 Kbps to 3 Mbps; (b) sum of two normal distributions with 300 Kbps average and 50Kbps standard deviation and with 3 Mbps average and 1 Mbps standard deviation. On the other hand, the total sum of bandwidths of LAN links connected to each MAN link was used as the bandwidth of the MAN link. 6 Gbps was used as bandwidths for WAN links.

In the above simulation configuration, we measured the average user satisfaction degree ($\frac{1}{|U|}\sum_{u\in U} S_u$, U is the set of all users). We changed the number of user nodes from 1 to 1000 and measured the average satisfaction degree for the combination of the above quality requirement distributions (a), (b) and four different types of populations of user nodes shown in Table 2. The experimental results are shown in Fig. 3, Fig. 4. In the figures, X-axis and Y-axis represent the number of nodes and the average satisfaction degree, respectively.

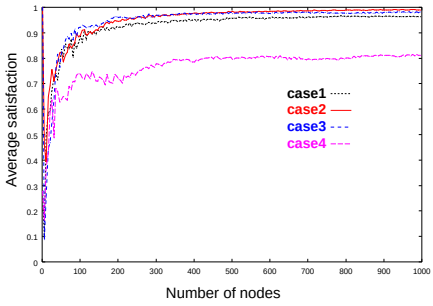

Fig. 3. Average User Satisfaction by requirement (a)

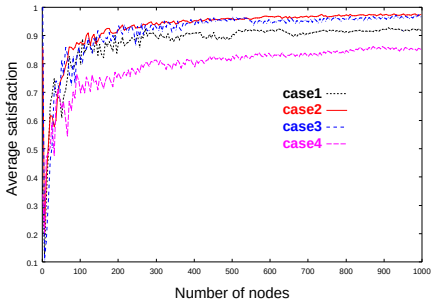

Fig. 4. Average User Satisfaction by requirement (b)

From Fig. 3, Fig. 4 , we see that MTcast can achieve pretty high satisfaction degree for various distribution of quality requirements from user nodes, when the number of user nodes are more than 100. The satisfaction degree is lower in case 4 than other cases.

This is because the percentage of user nodes with higher bandwidth is much smaller in case 4. However, even in such a case, MTcast achieved more than 70% user satisfaction.

In order to measure variation of user satisfaction degree depending on the value of k, we measured average user satisfaction degrees for $k = 2, 3, 6$ and 9 which are derived when applying four different combinations of $u.n_{trans}(u.q)$ and $u.n_{link}(u.q)$ in Table 3. From Table 3, when $k = 2$ or $k = 3$, the system can be recovered from one node failure per layer, and when $k = 6$ or $k = 9$, the system can be recovered from two and three simultaneous node failures per layer, respectively (these are calculated by equation (3). However, as the value of k increases, the average user satisfaction degree might decrease since the delivered quality is averaged among k members of each layer. The experimental result is shown in Fig. 5.

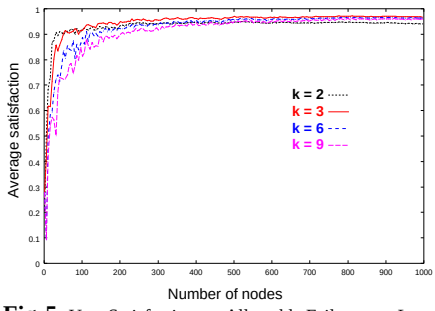

Fig. 5. User Satisfaction vs. Allowable Failures per Layer

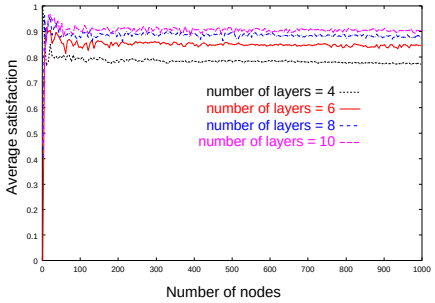

Fig. 6. Average User Satisfaction by Layered Multicast

From Fig. 5, while the number of nodes is relatively small (i.e., less than 300), the average user satisfaction degree decreases as the value of k increases. However, as the number of user nodes increases, the decrease gets smaller. From the result, while the number of user nodes is small, we should keep the value of k small in order to keep the average user satisfaction degree high, and we should increase the value of k gradually to improve robustness against node failure as the number of users increases.

For comparison, we also measured the average user satisfaction degree when using the layered multicast method. The average user satisfaction degree depends largely on the proportion of bitrates among multiple layers. So, we used the following way for allocating bitrates of layers: The average user satisfaction degree was considered as the evaluation function, and the optimal allocation of encoding rates were calculated for basic and extension layers using the Simulated Annealing method (the number of repetition times were 10,000).

With this optimization technique, we measured the average user satisfaction degrees. The results are shown in Fig. 6. From Fig. 5 and Fig. 6, when the number of user nodes is small (less than 200), and we use $k = 6$ or $k = 9$ with MTcast for two failure recovery per layer, the layered multicast achieves higher satisfaction degree than MTcast. when the number of nodes is sufficient (more than 200), MTcast achieves much higher satisfaction degree than the layered multicast with less than 10 layers (when the number of layers is higher than 10, the computational complexity will exceed the power of an ordinary PC [3]).

5 Concluding Remarks

In this paper, we proposed a new video delivery method called MTcast to achieve efficient simultaneous video delivery to multiple heterogeneous users. In the proposed

method, the same video stream is transmitted from a video server to user nodes by step-by-step transcoding at each intermediate node. The main contributions of MTcast are the following: (1) quick failure recovery and new user's quick reception of video streams can be achieved owing to layers of user nodes, (2) the size and height of the tree are kept small by periodical tree reconstruction, and (3) higher user satisfaction can be achieved with reasonable resource consumption at user nodes.

The above (2) also allows users to play back video segments with various different quality. When we use MTcast with our energy consumption control technique in [11], users can increase playback quality for preferred video segments without shortening playable time at portable devices within the battery amount.

In this paper, we only provided a centralized algorithm for constructing the transcode tree, although it works for the scale of 100,000 nodes. As part of future work, we want to design a distributed algorithm for tree construction to improve scalability further.

References

1. G.J. Conklin, G.S. Greenbaum, K.O. Lillevold, A.F. Lippman, and Y.A. Reznik, "Video Coding for Streaming Media Delivery on the Internet," IEEE Trans. on Circuits and Systems for Video Technology, Vol. 11, No. 3, pp. 269–281, 2001.
2. S. Jacobs and A. Eleftheriadis, "Streaming Video using Dynamic Rate Shaping and TCP Flow Control," Visual Communication and Image Representation Journal, 1998.
3. J. Liu, B. Li, and Y. Zhang, "An End-to-End Adaptation Protocol for Layered Video Multicast Using Optimal Rate Allocation," IEEE Trans. on Multimedia, Vol. 6, No. 1, pp. 87–102, 2004.
4. B. Vickers, C. Albuquerque, and T. Suda, "Source-Adaptive Multilayered Multicast Algorithms for Real-Time Video Distribution," IEEE/ACM Trans. on Networking, Vol. 8, No. 6, pp. 720–733, 2000.
5. H. Radha, M. van der Schaar, and Y. Chen, "The MPEG-4 Fine-Grained-Scalable video coding method for multimedia streaming over IP," IEEE Trans. on Multimedia, Vol. 3, No. 1, pp. 53–68, 2001.
6. S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications," Proc. of IEEE Infocom 2003, pp. 1521–1531, 2003.
7. V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," Proc. of the 12th Int'l. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2002), pp. 177–186, 2002.
8. J. Winick and S. Jamin, "Inet-3.0: Internet Topology Generator," Tech. Report UM-CSE-TR-456-02 (http://irl.eecs.umich.edu/jamin/), 2002.
9. Shuichi Yamaoka, Tao Sun, Morihiko Tamai, Keiichi Yasumoto, Naoki Shibata, and Minoru Ito, "ResourceAware Service Composition for Video Multicast to Heterogeneous Mobile Users" to appear in 1st International Workshop on Multimedia Service Composition, 2005.
10. R.S. Prasad, M. Murray, C. Dovrolis, and K.C. Claffy, "Bandwidth Estimation: Metrics, Measurement Techniques, and Tools," IEEE Network, Vol. 17, No. 6, pp. 27–35, 2003.
11. M. Tamai, T. Sun, K. Yasumoto, N. Shibata, and M. Ito, "Energy-aware Video Streaming with QoS Control for Portable Computing Devices," Proc. of the 14th Int'l. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2004), pp. 68–73, 2004.